Rapport de spécifications et conceptions du TP n°1 de Programmation Concurrente

Adrien Lepic, Marcin Konefal et Quentin Vecchio (B3323)

INSA Lyon - 3IF

Vendredi 8 Avril 2016

I - Introduction

Durant ce TP de programmation concurrente, il nous était demandé de développer une application permettant de gérer un parking. L'application devait être capable de faire rentrer des voitures selon certaines priorités (Un usager de type *PROF* est prioritaire sur un usager de type *AUTRE*) et également de devoir les faire sortir. Il fallait également gérer les usagers qui voulait même si le parking était plein. Pour finir tout cela devait être géré simultanément. Suite à notre conception générale (cf : rendu de la première séance), nous avons décidé de diviser l'application en 4 tâches.

II - Structures de communication

Le système Unix nous donne plusieurs moyens pour faire de la communication entres nos processus. Cependant il est important de déclarer des types spéciaux pour la communication. De ce fait nous avons créer plusieurs structures :

A - Structure Voiture

Elle contient toutes les informations concernant une voiture, notamment le numéro de la place qu'elle occupe (-1 si elle n'en occupe pas), le numéro de sa plaque (qui est compris entre 0 et 999), le type d'usager (de type TypeUsager), et le timestamp correspondant au moment où il est arrivé.

B - Structure StructDemandeSortie

Elle contient un champ type qui permet de l'identifier quand on met la structure dans une boite aux lettres, et un entier qui correspondant au numéro de la place que l'on veut libérer.

C - Structure StructRequete

Elle contient un champ type qui permet de l'identifier quand on met la structure dans une boite aux lettres, ainsi qu'un attribut de type *Voiture* et un autre de type *TypeBarriere*. Cette structure nous permet de définir une requête d'entrer dans le parking.

D - Structure StructTabRequetes

Elle contient un champ type qui permet de l'identifier quand on met la structure dans la mémoire partagée, et un tableau de *Requete*. Cette structure nous permet de mettre en place la mémoire partagée qui stocke les voitures en attente d'entrer dans le parking.

E - Structure StructParking

Elle contient un champ type qui permet de l'identifier quand on met la structure dans la mémoire partagée, et un tableau de *Requete*. Cette structure nous permet de mettre en place la mémoire partagée qui stocke les voitures en attente d'entrer dans le parking.

III - Moyens de communication

Pour faire communiquer nos tâches nous avons mis en place 4 boites aux lettres, 3 mémoires partagées et 6 sémaphores (3 de synchronisation et 3 pour gérer les accès concurrents sur les mémoires partagées)

Contient des messages de type StructRequete

Contient des messages de type StructDemandeSortie

Contient une variable de type unsigned int

Contient une variable de type StructTabRequetes

Contient une variable de type StructParking

IV - Tâche Mère

La première tâche est la tâche *Mère*. Elle devra, dans sa phase d'initialisation, créer les différents moyens de communications et les initialiser pour les autres tâches (Mémoires partagés, Boites aux lettres), mais aussi créer toutes les sémaphores pour gérer les accès concurrents. Elle doit également lancer les autres tâches. Dans sa phase moteur, la mère attend que la tâche *GestionClavier* ait terminée son execution. Dans sa phase destruction, la tâche mère doit veiller à détruire toutes les tâches (en envoyant à chacune le signal SIGUSR2), mais aussi à détruire tous les moyens de communications et sémaphores mis en place dans la phase d'initialisation.

V - Tâche GestionClavier (Simulateur)

```
void GestionClavier(int * BalEntreesId, int BalSortieId);
```

La tâche *GestionClavier* a pour but de gérer les interactions de l'utilisateur avec l'application. Les librairies misent en place par nos professeurs font appel à la fonction *Commande* déclarée dans cette tâche. Cette tâche ne fait pas d'initialisation particulière. Dans sa phase moteur la tâche fait appel à la fonction *Menu*. À la fin de l'execution de celle-ci, un appel à la fonction *Commande* est fait, la fonction se charge de déposer une requête dans la boite aux lettres (Boite aux lettres des entrées (1 à 3) ou la boite aux lettres de la sortie) correspondant à la demande de l'utilisateur. Si l'utilisateur demande à quitter l'application, alors la fonction *Commande* met fin à la tâche *GestionClavier*.

VI - Tâche Entrée

```
void Entree(int numero, int idBal, int idSemaphoreSynchro,
            int idSemaphoreContenuParking,
            int idSemaphoreRequete,
            int idSemaphoreNbPlaces ,
            int idMPContenuParking,
            int idMPRequete,
            int idMPNbPlaces);
```

La tâche *Entrée* a pour but de gérer les demandes d'entrée dans le parking. Dans sa phase d'initialisation la tâche attache les différentes mémoires partagées. Elle va également déclarer deux handlers, le premier qui va permettre d'intercepter le signal *SIGUSR2* pour détruire la tâche, et le second qui va permettre de récupérer le signal *SIGCHLD* pour gérer la fin des *Voituriers* (tâches filles). Nous avons également de besoin de créer un tableau contenant tous les pids des tâches lancées. Dans sa phase de moteur, la tâche attend qu'une requête soit disponible dans sa boite aux lettres. Une fois une requête disponible, la tâche regarde dans la mémoire partagée si de la place est disponible dans le parking, si c'est le cas alors une tâche fille (*Voiturier*) est lancée. Si ce n'est pas le cas, alors une requête est placée dans la mémoire partagée et on attend (avec un sémaphore de synchronisation) que la *Sortie* nous donne la permission de faire entrer la voiture dans le parking.

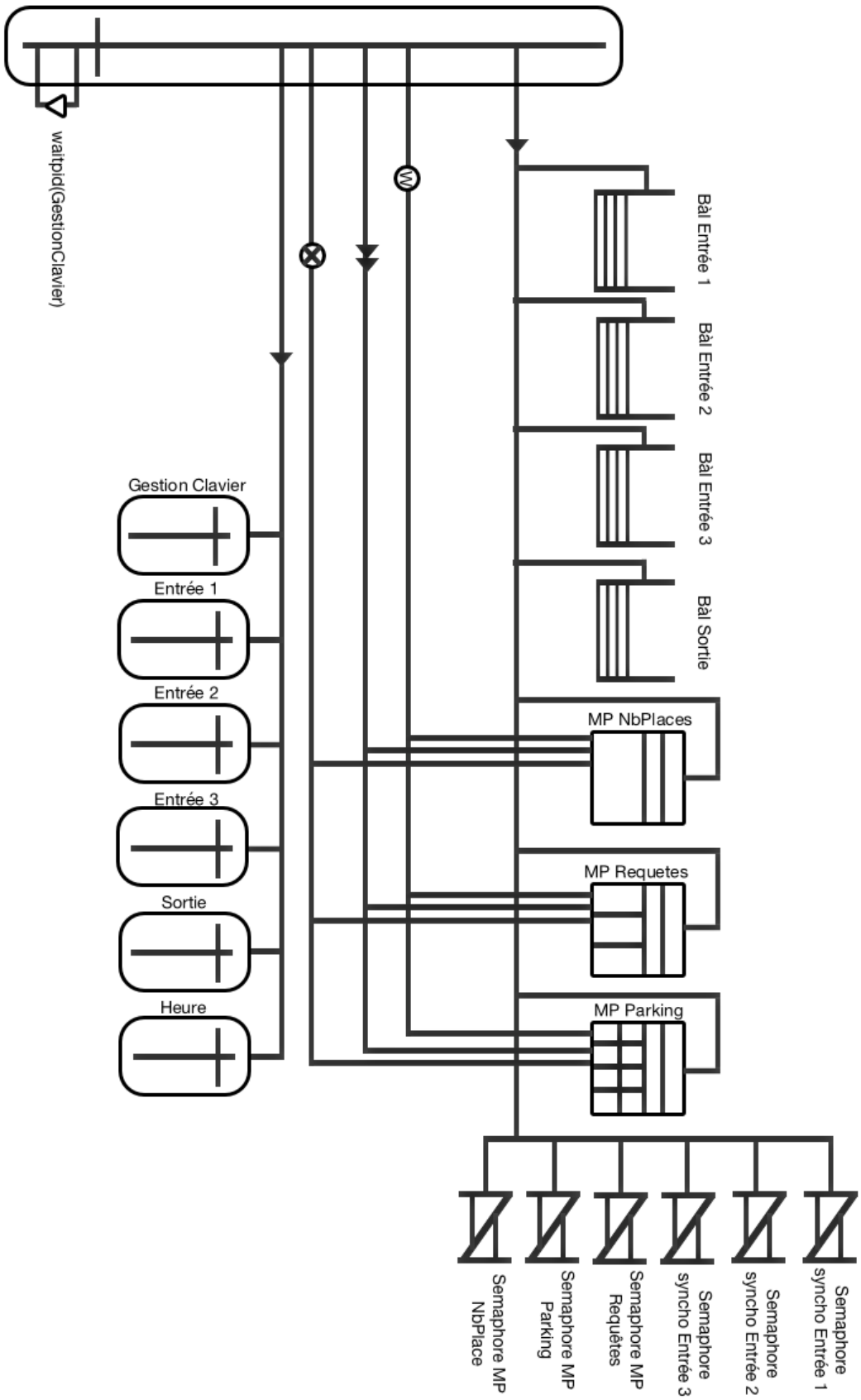

Schéma 1 : Conception tâche *Mère* (Phase initialisation et moteur)

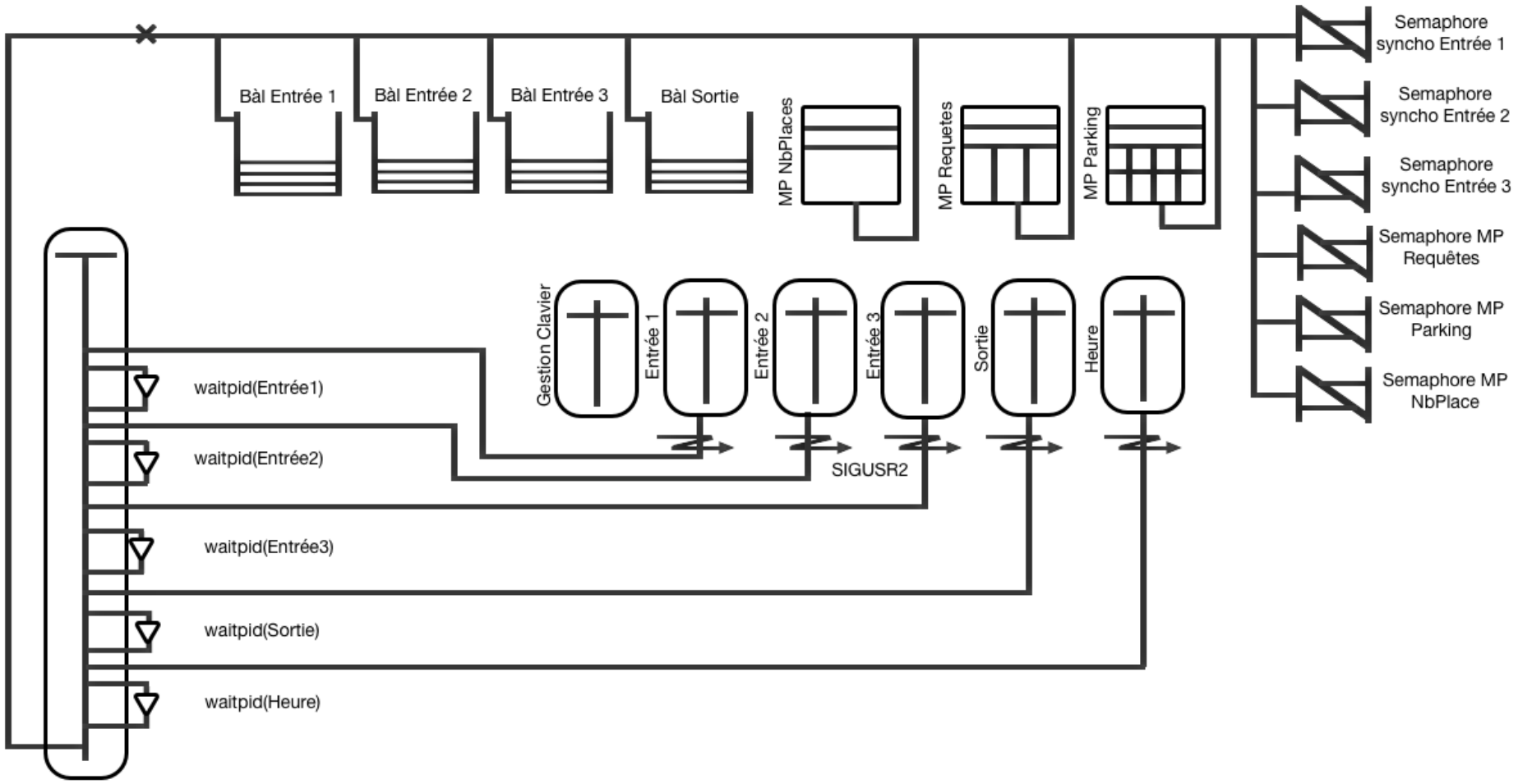

Schéma 2 : Conception tâche *Mère* (Phase destruction)

Schéma 3 : Conception tâche *GestionClavier*

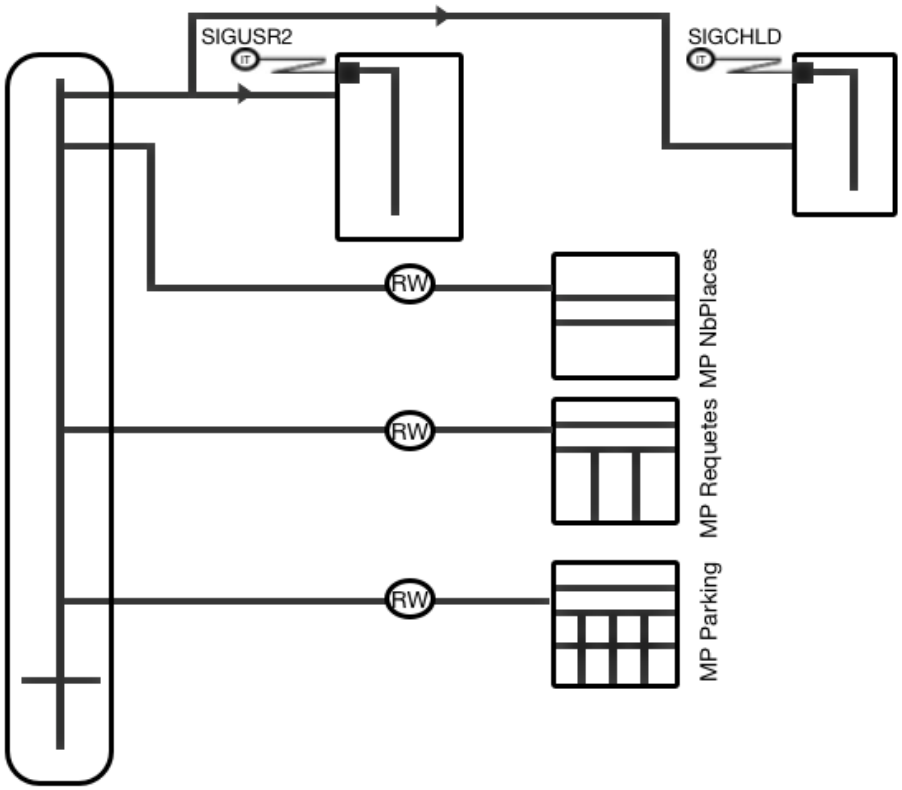

Schéma 4 : Conception tâche *Entree* (Phase initialisation)

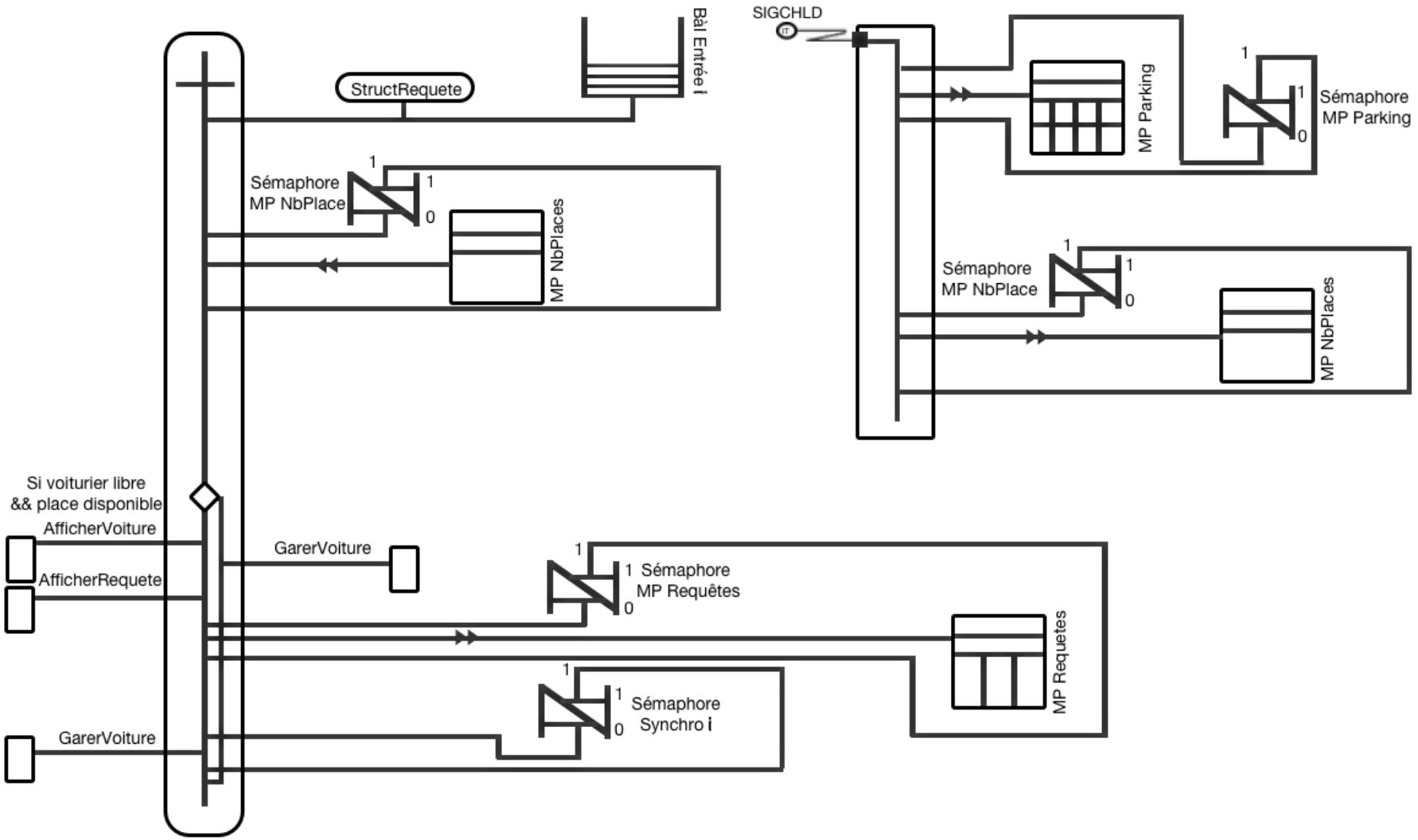

Schéma 5 : Conception tâche *Entree* (Phase moteur)

Schéma 6 : Conception tâche *Entree* (Phase destruction)

VII - Tâche Sortie

```
void Sortie(int idBal, int* idSemaphoreSynchro,
            int idSemaphoreContenuParking,
            int idSemaphoreRequete,
            int idSemaphoreNbPlaces ,
            int idMPContenuParking,
            int idMPRequete,
            int idMPNbPlaces);
```

La tâche *Sortie* a pour but de gérer les demandes de sortie du parking. Dans sa phase d'initialisation la tâche attache les différentes mémoires partagées. Elle va également déclarer deux handlers, le premier qui va permettre d'intercepter le signal *SIGUSR2* pour détruire la tâche, et le second qui va permettre de récupérer le signal *SIGCHLD* pour gérer la fin des *Voituriers* (tâches filles). Nous avons également de besoin de créer un tableau contenant tous les pids des tâches lancées. Dans sa phase de moteur, la tâche attend qu'une requête de sortie soit disponible dans sa boite aux lettres. Une fois une requête disponible, la tâche regarde dans la mémoire partagée si la place donnée est occupée dans le parking, si c'est le cas alors une tâche fille (*Voiturier*) est lancée. Une fois la voiture sortie, la tâche va consulter la table des requêtes pour voir si une voiture est en attente d'autorisation pour entrer dans le parking. Si une voiture est disponible et est prioritaire pour entrer alors la sémaphore de synchronisation est mise à 1 pour que la voiture entre.

VIII - Changements apportés

Au cours du développement de l'application nous nous sommes rendu compte de certaines erreurs sur notre conception générale. En effet, il nous manquait une mémoire partagée qui contenait les details du parking, ainsi qu'un mutex pour protéger cette mémoire. De plus nous avions mis en place, dans la tâche *GestionClavier* (Simulateur) un envoie du signal *SIGUSR2* à la tâche *Mère* pour lui signaler la fin de l'application. Nous avons remplacer ce système par une attente de fin de tâche (de *GestionClavier*) dans la phase moteur de la tâche *Mère*.

IV - Conclusion

Afin de finir dans les temps cette application, et en nous évitant des heures de travail inutile à la recherche d'anomalies, nous avons travaillé pas à pas en testant l'application à chaque ajout de fonctionnalités.

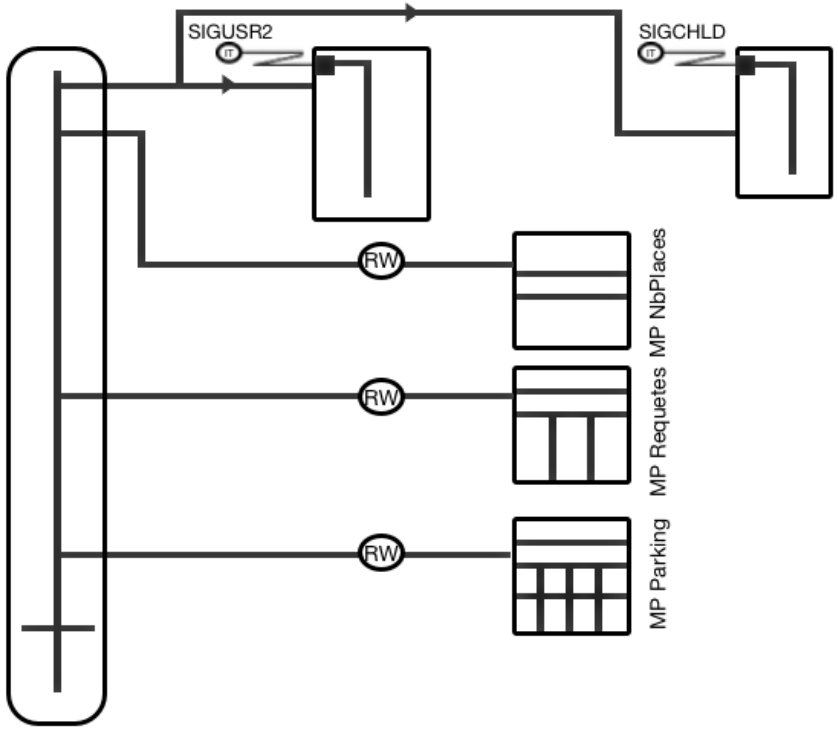

Schéma 7 : Conception tâche *Sortie* (Phase initialisation)

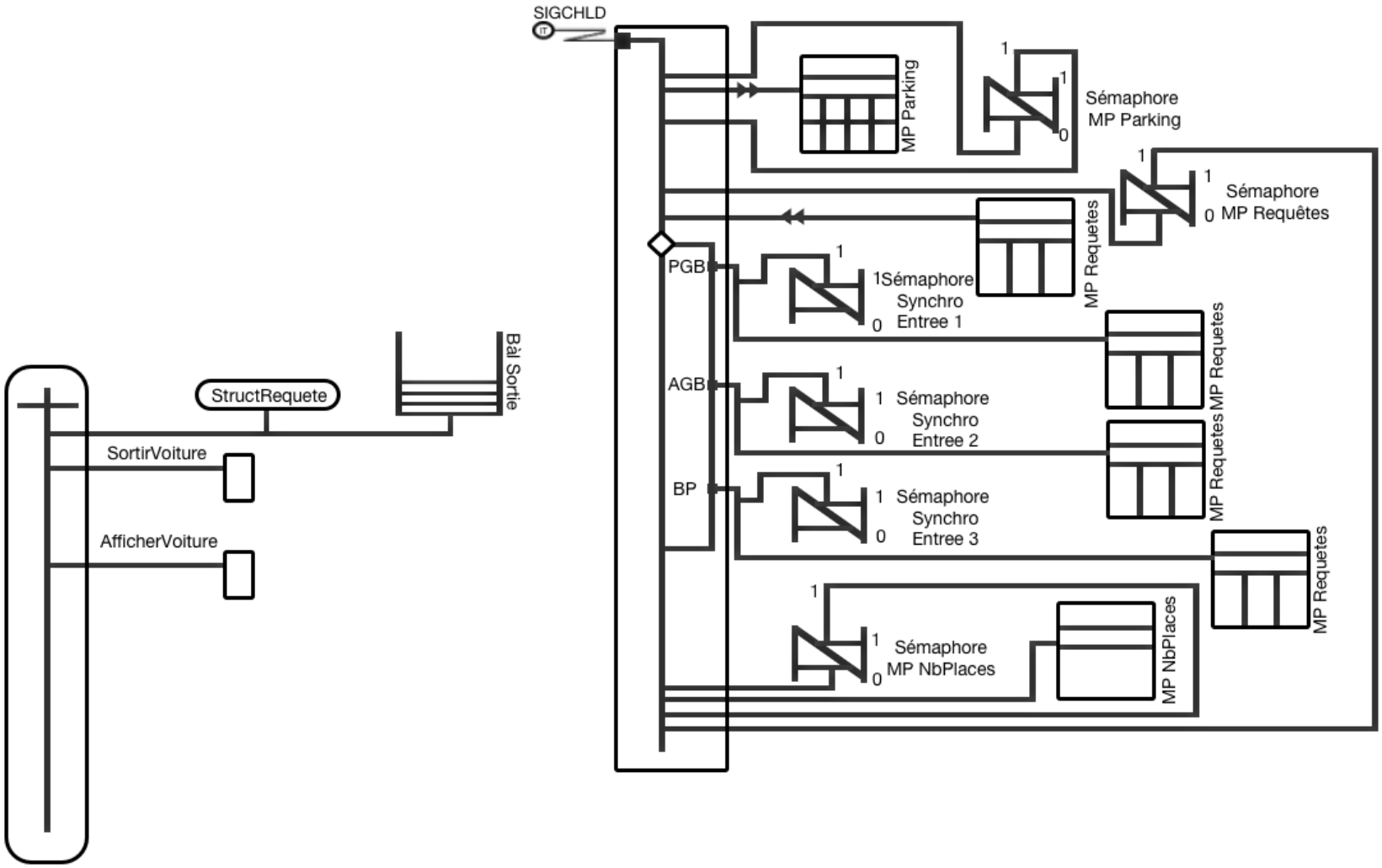

Schéma 8 : Conception tâche *Sortie* (Phase moteur)

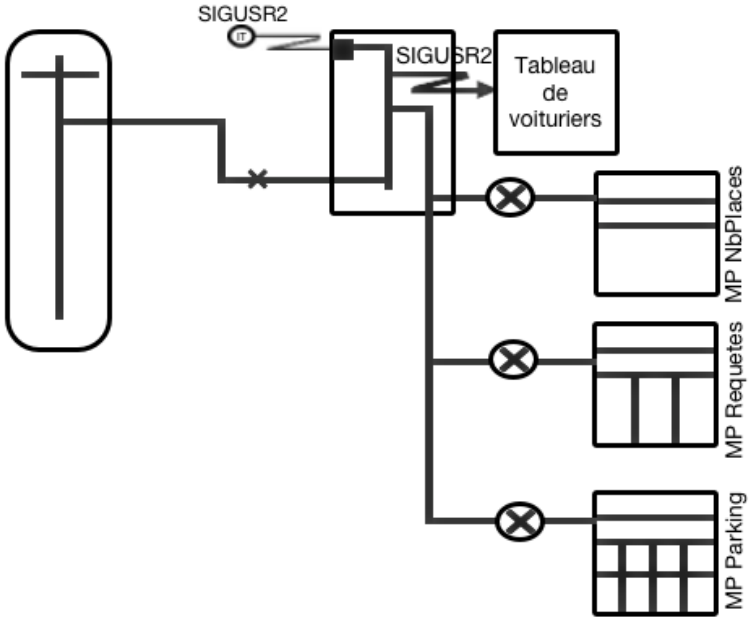

Schéma 9 : Conception tâche *Sortie* (Phase destruction)